FPGA-based Real-Time Pedestrian Detection on High-Resolution Images

Conference Paper · June 2013

DOI: 10.1109/CVPRW.2013.95

Michael Hahnle
Fachhochschule Aschaffenburg
2 PUBLICATIONS **31** CITATIONS

Frerk Saxen
Otto-von-Guericke-Universität Magdeburg
13 PUBLICATIONS **38** CITATIONS

Ulrich Brunsmann
Fachhochschule Aschaffenburg
34 PUBLICATIONS **326** CITATIONS

K. Doll
Fachhochschule Aschaffenburg
23 PUBLICATIONS **668** CITATIONS

FPGA-based Real-Time Pedestrian Detection on High-Resolution Images

Michael Hahnle[1], Frerk Saxen[2], Matthias Hisung[1], Ulrich Brunsmann[1], Konrad Doll[1]

[1]University of Applied Sciences Aschaffenburg, Germany
[2]Otto-von-Guericke University Magdeburg, Germany
{michael.hahnle, matthias.hisung, ulrich.brunsmann, konrad.doll}@h-ab.de
frerk.saxen@ovgu.de

Abstract

This paper focuses on real-time pedestrian detection on Field Programmable Gate Arrays (FPGAs) using the Histograms of Oriented Gradients (HOG) descriptor in combination with a Support Vector Machine (SVM) for classification as a basic method. We propose to process image data at twice the pixel frequency and to normalize blocks with the L1-Sqrt-norm resulting in an efficient resource utilization. This implementation allows for parallel computation of different scales. Combined with a time-multiplex approach we increase multiscale capabilities beyond resource limitations. We are able to process 64 *high resolution images* (1920×1080 *pixels) per second at* 18 *scales with a latency of less than* 150 μs. 1.79 *million HOG descriptors and their SVM classifications can be calculated per second and per scale, which outperforms current FPGA implementations by a factor of* 4.

1. INTRODUCTION

1.1. Motivation

A wide field of applications, *e.g.* surveillance systems, traffic assistance systems, autonomous robot navigation, etc., drive enormous research efforts in pedestrian detection. Satisfactory classification performance and speed is still not reached. Most systems are based on monocular vision. More and more applications ask for high resolution cameras to cover a wide field of view and resolve far distant objects. We concentrate on infrastructure based traffic assistance systems, which address pedestrian perception aiming at an improvement of road safety [8, 12] , but the concepts presented in this paper are applicable to most other applications, too.

The HOG descriptor together with linear SVM classification has shown good performance at a reasonable computation speed [6, 7]. To detect pedestrians of different size and different distance, the images are downscaled several times in practice. Real-time pedestrian detection is still a challenge especially for high resolution images. Parallel computing hardware like FPGAs is often used to provide satisfactory computation speed for real-time performance.

1.2. Related Research

Several pedestrian detection systems which use HOG or similar descriptors running on an FPGA platform have been described in literature. Some of them are based on a combined FPGA and Graphical Processing Unit (GPU) platform, *e.g.* Bauer *et al.* [1]. A CPU core embedded in an FPGA is used by Brookshire *et al.* [3]. Parts of the descriptor are evaluated on the FPGA, the rest of the descriptor and the classification is done on the CPU core. Most systems are purely FPGA based: Kadota *et al.* [10] concentrate on HOG feature extraction without implementing classification and propose several methods to simplify operations like *e.g.* division and square root. Binary patterned HOG features and an AdaBoost classifier were presented by Negi *et al.* [16]. Martelli *et al.* [14] and Hiromoto and Miyamoto [9] also use the detection window approach with cells and blocks but extract different features like covariance matrices or co-occurrence HOGs. A cell-based pipelining approach to calculate the HOG descriptor and a simultaneous SVM evaluation is described in Mizuno *et al.* [15].

The maximum resolution of the above systems experimentally demonstrated on real FPGA hardware is 800×600 pixels at 72 fps [15]. The same authors mention that their system is expandable to HD resolution videos at 30 fps. In addition, all systems use the original image size, downscaling is not implemented, and they apply the L2-norm, the L2-Hys-norm,or approximations of them, if at all, for block normalization.

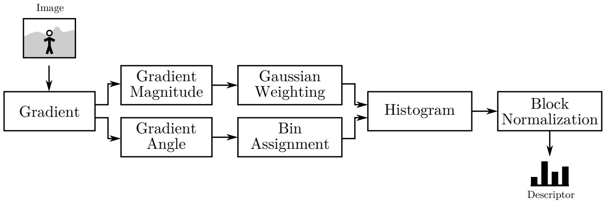

Figure 1. Calculation steps of the HOG descriptor

1.3. Our Contribution

The main contributions of this paper are: (1) We process data at twice the pixel frequency and (2) normalize HOG blocks with the L1-Sqrt-norm. Both aspects result in a very efficient resource utilization and high throughput. (3) The efficient resource utilization allows for several parallel instantiations of processing pipelines for multiscale pedestrian detection. We propose a time-multiplex method resulting in the computation of more scalings than allowed by the resource limits. (4) Experimental results show that the system runs with HD resolution images at 64 fps at 18 scales.

1.4. Overview

The paper is organized as follows: In Section 2 we explain the basics of our HOG and SVM based algorithm. The hardware architecture and the details of the implementation are described in Section 3. In Section 4 we present the experimental results before we summarize the main conclusions and discuss open issues in Section 5.

2. ALGORITHM

Our method is based on the HOG descriptor followed by an SVM classification as proposed by Dalal and Triggs [6]. Image based pedestrian detection is performed using a sliding window approach: By sliding a detection window of constant size (64×128 pixels) from top left to bottom right the entire image is analyzed for pedestrians.

The analysis consists of two parts: First, a descriptor is calculated for each detection window by using the HOG-algorithm. Afterwards, the descriptors are classified applying an SVM. In the following sections we describe the details of both parts putting emphasis on our implementation.

2.1. Histograms of Oriented Gradients Descriptor

Fig. 1 shows the steps necessary for computing the HOG descriptor. For each pixel (x, y) in the input image I the two-dimensional gradient $G(x, y) = (G_x(x, y), G_y(x, y))$ is determined. The gradient magnitude $|G(x, y)|$ and gradient angle $\phi(x, y)$ at position (x, y) are given by

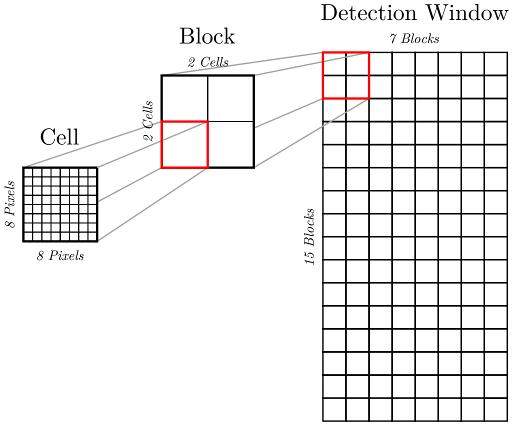

Figure 2. Division of the detection window into blocks and cells

$$|G(x, y)| = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}, \qquad (1)$$

$$\tan(\phi(x, y)) = \frac{G_y(x, y)}{G_x(x, y)}. \qquad (2)$$

The detection window is divided into non overlapping cells (Fig. 2) with a size of 8×8 pixels. Four neighbouring cells form a block. The blocks have a horizontal and vertical overlap of one cell. This results in $7 \cdot 15 = 105$ blocks in a detection window.

For each cell a histogram is generated by defining 9 bins for the gradient angle and accumulating weighted gradient magnitudes in each of the bins. Concatenating the histograms of cells contained in a block delivers a block histogram with $4 \cdot 9 = 36$ elements.

To improve detection quality, the gradient magnitudes at the boundary area of the blocks are weighted less than the centered ones. Hence, the gradient magnitudes in a block $|G|_{Block}$ are multiplied with a 16×16 sized Gaussian matrix F_g

$$|G|_{Gaussian} = F_g \circ |G|_{Block}, \qquad (3)$$

where $\circ$ denotes the multiplication element by element and $\sigma = 8$ [6].

The weighted magnitudes are added per bin to get the histogram elements.

For better invariance to illumination and contrast changes the block histograms $\underline{v}$ are normalized.

We use the L1-Sqrt-norm

$$\underline{v} \rightarrow \sqrt{\frac{\underline{v}}{\|\underline{v}\|_1 + \epsilon}}. \qquad (4)$$

This, compared to the L2-norm, avoids squaring the histogram elements leading to an efficient implementation (see Sec. 3.3) without reducing the detection rate significantly (see Fig. 4 in [6]).

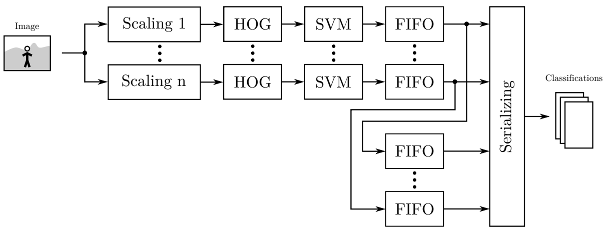

Figure 3. System overview with multiple scales and additional FIFOs for the time-multiplex extension

Finally, all 105 block histograms are concatenated to get the $105 \cdot 36 = 3780$-dimensional HOG descriptor $\underline{x}$.

2.2. Classification with a Support Vector Machine

For classification of the HOG descriptors we are using a linear SVM, which evaluates

$$y(\underline{x}) = \underline{w}^T \cdot \underline{x} + b. \tag{5}$$

The weight vector $\underline{w}$ and the bias b are determined during the training phase of the SVM. If $y(\underline{x}) > 0$, a pedestrian is detected, otherwise no pedestrian is detected.

3. IMPLEMENTATION

Our implementation uses a cell based approach similar to Mizuno *et al.* [15]: instead of calculating the entire descriptor for all detection windows one after another, we create all cell and block histograms in a first step. Since a block is part of 105 detection windows, we avoid multiple histogram calculations for one block. In a second step the detection window descriptors are composed of them. This is an essential aspect of our real-time FPGA implementation with high throughput.

The design is highly modularized to increase the flexibility for configuration and extension. In this chapter we describe the fundamental modules in detail with respect to their functionality and implementation.

3.1. System Overview

Fig. 3 shows the complete design including the multiscale and time-multiplex extension. Each scale consists of a scaling module, a HOG module for the calculation of the descriptors, and an SVM module for classification. The FIFOs and the serialization module are introduced to convert the classification outputs into the desired data format for further processing.

3.2. Scaling

The input image is downscaled by using bilinear interpolation because several scales significantly improve the detection rate in practice.

3.3. Descriptor Calculation (HOG)

Fig. 4 shows the structure of our HOG descriptor implementation. The calculation steps are labelled according to Fig. 1. The details are described below.

3.3.1 Gradient

The Gradient module consists of a row buffer, a shift register and two subtractors for parallel calculation of the horizontal, and vertical gradient components G_x and G_y.

Afterwards G_x and G_y are transferred to the core clock domain via the clock domain crossing module. The core clock frequency is twice the pixel clock frequency. This frequency doubling results in an enormous resource reduction for the following modules because one operator can process two operations.

3.3.2 Gradient Magnitude and Bin Assignment

For calculating the gradient magnitude G_x and G_y are squared and added. From this sum the square root is extracted. By serializing the multiplication one multiplier can be saved.

To avoid a resource expensive division for determining $\tan(\phi(x, y))$, the angle calculation is combined with the bin assignment [1, 4]. The assignment of a gradient to a bin can be done by multiplying $|G_x|$ with constant values and comparing it with $|G_y|$. *E.g.*, G is assigned to Bin 1 (0° - 20°) if

$$0 \leq \frac{|G_y|}{|G_x|} < \tan(20°). \tag{6}$$

We multiply with $|G_x|$ and get

$$\tan(0°) \leq |G_y| < \tan(20°) \cdot |G_x|. \tag{7}$$

If this inequality holds, the gradient gets assigned to Bin 1. To avoid floating point arithmetic in the FPGA implementation, the constants can be approximated by a fraction

$$\tan(20°) \approx \frac{4}{11}. \tag{8}$$

Now the inequality can be written using only integer values:

$$0 \leq |G_y| \cdot 11 < |G_x| \cdot 4. \tag{9}$$

For every bin a similar inequality has to be checked for the gradient assignment. Again, the doubled core clock frequency allows for serializing multiplications and results in half the multipliers.

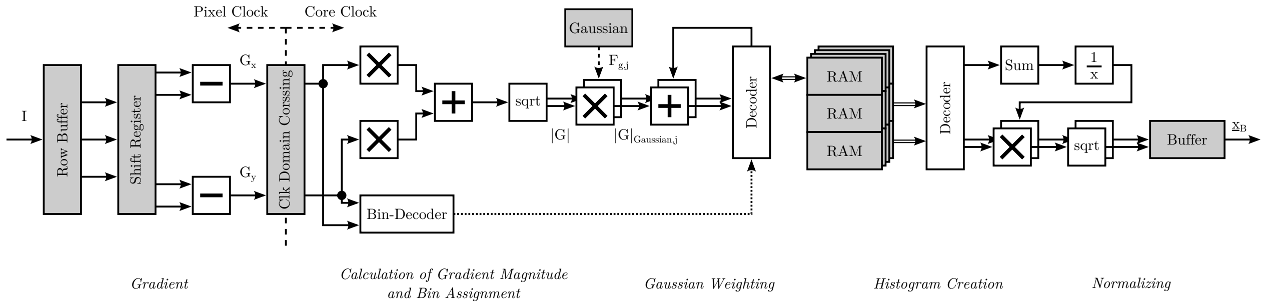

Figure 4. Structure of the HOG implementation

3.3.3 Gaussian Weighting

The Gaussian weighting of the gradient magnitudes within a block is achieved by multiplying them with precalculated values of a Gaussian filter matrix saved in a ROM. Because every cell is part of four overlapping blocks with a different position within these blocks, each cell is multiplied element by element with the four 8×8 pixel sized parts F_{g1} - F_{g4} of the Gaussian filter matrix

$$F_g = \left[\begin{array}{cc} F_{g1} & F_{g2} \\ F_{g3} & F_{g4} \end{array} \right], \tag{10}$$

$$\forall j \in \{1,2,3,4\} : |G|_{Gaussian,j} = F_{gj} \circ |G|_{Cell}. \tag{11}$$

The proposed method results in 4 multiplications per pixel clock cycle or 2 multiplications per core clock cycle.

3.3.4 Block Histogram Creation

To create a cell histogram, the weighted gradient magnitudes are added for each bin. Then four cell histograms are concatenated to get the final block histogram.

The calculation of the cell histograms is processed over the whole image width. Three lines of cell histograms are stored in a RAM module – two for the output of the block histograms and a third one as current input line where the input data is is accumulated.

The input decoder (left to the RAM module in Fig. 4 generates the read and write addresses for the RAM module and controls the memory access. The output decoder (right to the RAM module in Fig. 4) sequentially reads the four corresponding cell histograms for each block from the memory and sends it to the normalization module. It is also in charge of erasing the memory cells after the cell histograms have been read.

3.3.5 Normalizing

To obtain the L1-Sqrt-norm (see Eqn. 4), we calculate the sum of all elements of the four cell histograms $\|\underline{v}\|_1$ plus ϵ (we choose ϵ to be the smallest positive value), which takes 36 clock cycles.

Instead of dividing every element by $(\|\underline{v}\|_1 + \epsilon)$, we multiply each element with the inverse:

$$\underline{v} \cdot \left(\frac{1}{\|\underline{v}\|_1 + \epsilon} \right). \tag{12}$$

The computation of the inverse has to be done only once per block which reduces its resources to approximately a quarter of a fully pipelined divider and requires another 33 clock cycles. In addition, we need 36 clock cycles to reinitialize the RAM module with zero values.

Every cell histogram is based on $8 \cdot 8 = 64$ pixels, which corresponds to 64 pixel clock cycles or 128 core clock cycles available for real-time data processing. This means that we have $128 - 36 - 33 - 36 = 23$ core clock cycles left for multiplying the 36 block histogram elements with the value ${}^{1}/_{\|\underline{v}\|_1+\epsilon}$. We perform two simultaneous memory read accesses and two multiplications. These operations require 18 clock cycles which satisfies the requirement of less than or equal 23 clock cycles.

After that the square root is extracted in a pipeline and the results are stored in an output buffer until the entire block histogram is normalized and $\underline{x}_B$ is available.

A similar analysis to L2-norm shows that the square root, which takes 7 clock cycles, has to be extracted before each element of the histogram can be multiplied by the inverse. This means that the number of clock cycles left for multiplying the elements are less than 18 and we would need more multipliers and more memory accesses making the design much more complex and resource consuming.

3.4. Classification (SVM)

The SVM evaluates the equation:

$$y(\underline{x}) = \underline{w}^T \cdot \underline{x} + b. \tag{13}$$

The weight vector $\underline{w}$ and the bias b are constant for all evaluations. $\underline{w}$ is stored in a ROM on the FPGA.

Eqn. 13 can be rewritten using the 105 block histograms

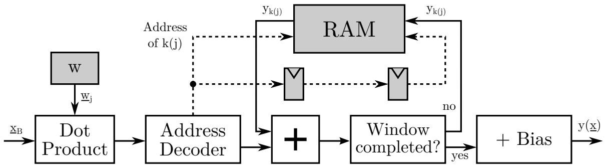

Figure 5. Structure of the SVM implementation

$\underline{x}_{Bi}$ belonging to a detection window:

$$y(\underline{x}) = \underline{w}^T \cdot \underline{x} + b = \sum_{i=1}^{105} \left(\underline{w}_i^T \cdot \underline{x}_{Bi} \right) + b, \tag{14}$$

with

$$\begin{aligned} \underline{x}^T &= \left[\ \underline{x}_{B1}^T \quad \underline{x}_{B2}^T \quad \cdots \quad \underline{x}_{B105}^T \ \right], & (15)\\ \underline{w}^T &= \left[\ \underline{w}_1^T \quad \underline{w}_2^T \quad \cdots \quad \underline{w}_{105}^T \ \right]. & (16) \end{aligned}$$

To evaluate $y(\underline{x})$ for one detection window, we need the histograms of all blocks belonging to it. To avoid the huge amount of memory needed for storing all the block histograms until the last block histogram of the detection window has been processed, we calculate portions $\underline{w}_i^T \cdot \underline{x}_{Bi}$ of the dot product and store the resulting scalars for each block as described in detail below:

We consider all detection windows overlapping a block. In each detection window and in the corresponding descriptor the block histogram $\underline{x}_B$ has a different position j and gets $\underline{x}_{B1}$ to $\underline{x}_{B105}$. The part of the vector $\underline{w}$ corresponding to position j is $\underline{w}_j$. The detection window in which $\underline{x}_B$ is located at position j is denoted as $k(j)$. A dot product has to be calculated of a block histogram $\underline{x}_B$ and each of the 105 parts of the vector $\underline{w}_1$ to $\underline{w}_{105}$ resulting in contributions to 105 detection windows. These dot products are then added to the current intermediate total $y_{k(j)}$ of the detection window $k(j)$:

$$\forall j \in \{1, \ldots, 105\} : y_{k(j)} \leftarrow y_{k(j)} + \underline{w}_j^T \cdot \underline{x}_B. \tag{17}$$

An overview of the hardware implementation following this approach is shown in Fig. 5. It contains the module for the dot product calculation, an address decoder handling the RAM accesses, an adder for the intermediate totals, and an adder for the bias. By checking the sign bit of $y(\underline{x})$, the class correspondence of the specific classification window can be determined.

The SVM outputs the position of the detection window in the unscaled input image, the width and height of the unscaled window and $y(\underline{x})$ for each positive pedestrian detection.

3.5. Multiscale Approach

To perform multiscale detection several processing pipelines are instantiated in parallel. Every scaling module is configured to provide a different scale. The positive detections from the outputs of the SVMs have to be serialized for transmission. A FIFO behind each SVM module buffers the detection results (Fig. 3). A serializing module requests detection data from the FIFOs in turn and converts them into the desired output format, *e.g.* a byte stream.

3.6. Time-Multiplex Approach

To process about 20 different scalings as usually done in applications, there either have to be many processing pipelines in parallel or the image has to be stored and looped multiple times through one pipeline using different scales. The first alternative requires a large amount of resources, which in many cases are not available, the second one conflicts with the given real-time requirements. We use a combination of both alternatives and develop it further to meet the resource and real-time requirements: At an image frequency of 50 fps there are 20 ms between two images. Considering the infrastructure based camera setup, even at a fast walking speed a pedestrian is moving only a few pixels from one image to the next. This means: If a pedestrian has been detected in an image, we can assume that the pedestrian is detected in the same detection window in the next image. Based on this we use different scales for detecting pedestrians in the next image. All detections of an image can be finally determined by combining the detections of the current and the last image. Experimental and theoretical analysis show that we can combine detections not only from two images but also from three images.

In our case we use 18 scales. Instead of processing every image with 18 scales in parallel requiring resources for all the scales, we process the first image with scales $1-6$, the second image with scales $7-12$, the third image with scales $13-18$, the forth image again with scales $1-6$ and so forth.

This time-multiplex approach is implemented by reconfiguring the scaling modules after each image. Thus, multiple scales can be processed with the same pipeline. To combine all scales for every image, the detection results of the last images are looped through a FIFO cascade whose length depends on the number of scalings. The output data of the single cascade steps are then also sent to the serializing module for output (Fig. 3).

4. EXPERIMENTAL RESULTS

Our test system consists of a Full-HD camera Prosilica GX1910[1] triggered to deliver 50 fps with 8 bit grayscale. The image data are transferred via GigE to an XpressGen2V5-200 FPGA Board[2] with a Xilinx Virtex®–5 FPGA (XC5VFX200T). A GigE Vision Core[3] receives

[1] http://www.alliedvisiontec.com/

[2] http://www.plda.com/

[3] http://www.s2i.org/

	Scaling	HOG	SVM
# of registers	1 861	3 642	1 534
# of LUTs	1 177	3 924	1 264
# of DSP blocks	8	12	37
memory (kBit)	36	936	252

Table 1. Resources for Scaling, HOG, and SVM modules

	Design	Available	Utilization
# of registers	42 987	122 880	34 %
# of LUTs	38 535	122 880	31 %
# of DSP blocks	357	384	92 %
memory (kBit)	7 128	16 416	43 %

Table 2. Resources and load for a Xilinx Virtex®–5 FPGA (XC5VFX200T) with 6 scales in parallel and 3 scale sets in time multiplex

the data and converts it to a pixel stream, which is analyzed by our pedestrian detection module. The results together with the input image are transferred by a PCIe Core[2] using DMA to a PC for further processing. Further information about the system setup and its components can be found in [8, 11].

4.1. Resources

Table 1 lists the resources for the scaling, HOG and SVM modules for one processing pipeline at scale 1 (unscaled Full-HD input image). Table 2 shows the resources and the resource utilization for the complete design.

Because the DSP blocks are the limiting factor, we tried to minimize them in all modules. Using the clock frequency doubling approach the number of DSP blocks was reduced significantly for the HOG calculation. Especially the normalization needs much less resources because 128 clock cycles are available instead of only 64, which allows the highly serialized calculations as described above.

A comparison to other FPGA implementations is presented in Table 3. Compared to the implementation from Mizuno *et al.* [15], which also targets a resolution of 1920 × 1080 pixels, we use significantly less LUTs, Registers and DSPs. We need about twice the memory than Mizuno *et al.* [15] but this is not the limiting factor in resource utilization of this system. Although we use only a small amount of resources, we achieve about twice the throughput (64 fps). Even if we compare our system to other implementations targeting 640 × 480 pixels, we use less resources in most cases at the highest frame rate.

4.2. Real-Time Capability

The implementation is optimized for a pixel clock frequency of 133 MHz and a core clock frequency of 266 MHz. The maximum frequency of the core clock is approximately 270 MHz. Hence, the maximum image frequency is 64 fps at a resolution of 1920 × 1080 pixels (Table 3). This was experimentally verified using video files transferred to the FPGA via PCIe. The table shows, that no other implementation achieves this frequency. At the first scale,

$$\left(\frac{1\,920}{8}-7\right)\cdot\left(\frac{1\,080}{8}-15\right)\cdot 64 = 1\,789\,440 \qquad (18)$$

classification windows are classified per second. We shift the detection window not pixel by pixel but cell by cell. The number of detection windows can be calculated by dividing the image width and height by the cell width 8. Because a detection window consists of 8×16 cells and no window exceeds the image boundaries, 7 detection windows in width and 15 in height have to be subtracted in Eqn. 18. That exceeds current implementations [2, 13, 15, 16] by more than a factor of 4.

The latency of the implementation varies depending on the scale. Hardware simulation shows a latency of about 150 μs for the unscaled image. Latencies for all other scales are shorter because the images are smaller.

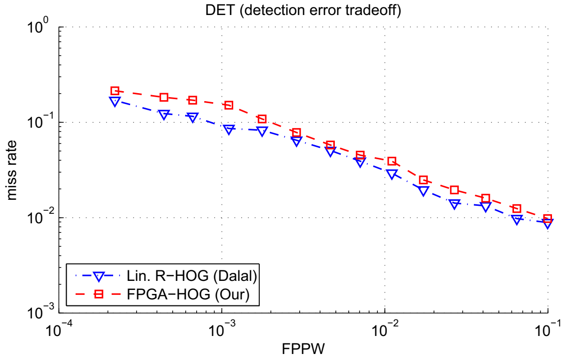

Figure 6. Comparison of the detection rate based on the INRIA person dataset[5]

4.3. Detection Rate

To evaluate the detection rate, the FPGA implementation was trained with the INRIA person dataset [5] and compared to a CPU implementation (original R-HOG [6] with a linear SVMlight-SVM). The results are shown in Fig. 6. There is a 6% higher miss rate at 10^{-3} FPPW than the R-HOG. Experiments with a Matlab reference model having the same behavior as the FPGA implementation show that this is mainly caused by omitting the trilinear interpolation during the histogram creation and the rounding of the calculations (especially within the normalization).

	Platform	Resolution	fps	LUTs	Registers	DSPs	Memory (kBit)	MHz	Windows / sec
[10]	Altera Stratix II	640 x 480	30	37 940	66 990	120	no data	127	56 466
[16]	Xilinx Virtex-5	320 x 240	62	17 383	2 181	no data	1 327	44	95 480
[15]	Altera Cyclone IV	800 x 600	72	34 403	23 247	68	348	76	401 760
Our	Xilinx Virtex-5	1920 x 1080	64	5 188	5 176	49	1 188	270	1 789 440

Table 3. Comparison of the resources for different FPGA implementations of one scale (resources of HOG + SVM from Table 1)

5. CONCLUSIONS

In this paper we presented a real-time FPGA implementation of a pedestrian detection system using a HOG descriptor and an SVM classification. Compared to other implementations [2, 13, 15, 16] a performance gain in the number of detection windows per second of more than a factor of 4 has been reached. Using a core clock twice as fast as the pixel clock and additional optimization methods, the resources have been significantly reduced without limiting the real-time capabilities of the design. Based on the time-multiplex approach we use 18 scales which are regarded to be necessary in practical applications.

Future research will concentrate on further optimizations of hardware resources. Other objectives are quality improvement and the integration of a Non Maxima Suppression into the FPGA.

6. ACKNOWLEDGEMENT

The support of the *Bayerisches Staatsministerium für Wissenschaft, Forschung und Kunst* in the context of the *Forschungsschwerpunkt Intelligente Verkehrssicherheits- und Informationssysteme* at the University of Applied Sciences Aschaffenburg is gratefully acknowledged. This work partially results from the joint project Ko-PER, which is part of the project initiative Ko-FAS, and has been funded by the German *Bundesministerium für Wirtschaft und Technologie* (Federal Department of Commerce and Technology) under grant number 19S9022B.

References

[1] S. Bauer, U. Brunsmann, and S. Schlotterbeck-Macht. FPGA Implementation of a HOG-based Pedestrian Recognition System. *42. Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, pages 49–58, July 2009.

[2] S. Bauer, S. Köhler, K. Doll, and U. Brunsmann. FPGA-GPU Architecture for Kernel SVM Pedestrian Detection. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 61–68, June 2010.

[3] J. Brookshire, J. SteffJorgenensen, and J. Xiao. FPGA-based Pedestrian Detection, May 2010.

[4] T. P. Cao and G. Deng. Real-Time Vision-Based Stop Sign Detection System on FPGA. *2008 Digital Image Computing: Techniques and Applications (DICTA)*, pages 465–471, December 2008.

[5] N. Dalal and B. Triggs. INRIA Person Dataset, 2005.

[6] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, June 2005.

[7] M. Enzweiler and D. M. Gavrila. Monocular Pedestrian Detection: Survey and Experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2179–2195, December 2009.

[8] M. Goldhammer, E. Strigel, D. Meissner, U. Brunsmann, K. Doll, and K. Dietmayer. Cooperative Multi Sensor Network for Traffic Safety Applications at Intersections. *15th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1178–1183, September 2012.

[9] M. Hiromoto and R. Miyamoto. Hardware Architecture for High-Accuracy Real-Time Pedestrian Detection with CoHOG Features. *12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 894–899, September 2009.

[10] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura. Hardware Architecture for HOG Feature Extraction. *Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 1330–1333, Spetember 2009.

[11] J. Kempf, M. Schmitt, S. Bauer, U. Brunsmann, and K. Doll. Real-Time Processing of High-Resolution Image Streams using a Flexible FPGA Platform. *Proceedings of the Embedded World Conference*, February 2012.

[12] Ko-FAS. Forschungsinitiative Ko-FAS, 2012.

[13] K. Lillywhite, D.-J. Lee, and D. Zhang. Real-time Human Detection Using Histograms of Oriented Gradients on a GPU. *2009 Workshop on Applications of Computer Vision*, pages 1–6, December 2009.

[14] S. Martelli, D. Tosato, M. Cristani, and V. Murino. FPGA-Based Pedestrian Detection Using Array of Covariance Features. *2011 Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, pages 1–6, August 2011.

[15] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto. Architectural Study of HOG Feature Extraction Processor for Real-Time Object Detection. *2012 IEEE Workshop on Signal Processing Systems*, pages 197–202, October 2012.

[16] K. Negi, K. Dohi, Y. Shibata, and K. Oguri. Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm. *2011 International Conference on Field-Programmable Technology*, pages 1–8, December 2011.